Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

«ПРОЦЕДУРЫ, ФУНКЦИИ, ТРИГГЕРЫ В POSTGRESQL»

по дисциплине «Проектирование и реализация баз данных»

Обучающийся Баженов Алексей Антонович

Факультет прикладной информатики

Группа K3240

Направление подготовки 09.03.03 Прикладная информатика

Образовательная программа Мобильные и сетевые технологии 2025

Преподаватель Говорова Марина Михайловна

Санкт-Петербург

Введение

Цель работы: овладеть практическими создания и использования процедур, функций и триггеров в базе данных PostgreSQL.

Практическое задание:

1. Создать 3 процедуры для индивидуальной БД согласно варианту (часть 4 ЛР 2). Допустимо использование IN/OUT параметров. Допустимо создать авторские процедуры. (3 балла)
2. Создать триггеры для индивидуальной БД согласно варианту:
   Вариант 2.1. 3 триггера - 3 балла (min). Допустимо использовать триггеры логирования из практического занятия по функциям и триггерам.
   Вариант 2.2. 7 оригинальных триггеров - 7 баллов (max).

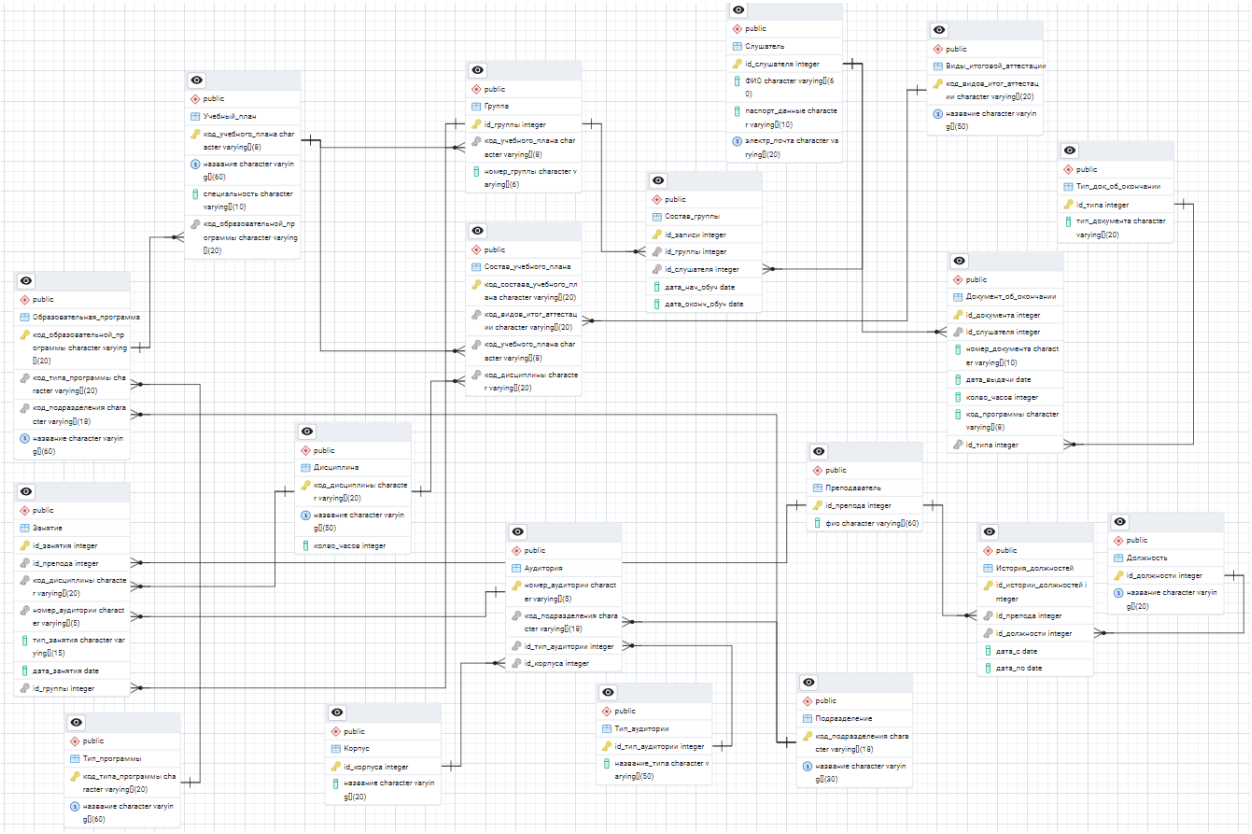

Рисунок 1 - Схема ИЛМ, сгенерированная в Generate ERD

Ход работы

Создать хранимые процедуры:

Для получения расписания занятий для групп на определенный день недели:

```
CREATE OR REPLACE FUNCTION get_group_schedule(
    group_id INT,
    day_of_week VARCHAR
)
RETURNS TABLE (
    id INT,
    date DATE,
    discipline VARCHAR(50),
    lesson_type VARCHAR(15),
    building VARCHAR(20),
    room VARCHAR(5),
    teacher VARCHAR(60),
    weekday TEXT
)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT
        з."id_занятия",
        з."дата_занятия",
        д."название",
        з."тип_занятия",
        к."название",
        а."номер_аудитории",
```

```
    п."фио",
    to_char(з."дата_занятия", 'Day')
  FROM "Занятие" з
  JOIN "Дисциплина" д ON з."код_дисциплины" = д."код_дисциплины"
  JOIN "Аудитория" а ON з."номер_аудитории" = а."номер_аудитории"
  JOIN "Корпус" к ON а."id_корпуса" = к."id_корпуса"
  JOIN "Преподаватель" п ON з."id_препода" = п."id_препода"
  WHERE з."id_группы" = group_id
    AND lower(to_char(з."дата_занятия", 'Day')) LIKE '%' ||
lower(trim(day_of_week)) || '%'
  ORDER BY з."дата_занятия";

  IF NOT FOUND THEN
    RAISE NOTICE 'Занятий для группы % в день "%" не найдено', group_id,
day_of_week;
  END IF;
END;
$$;
```

```
 id |    date    |       discipline        | lesson_type | building | room  |        teacher         |  weekday
----+------------+-------------------------+-------------+----------+-------+------------------------+-----------
  1 | 2025-04-11 | Основы программирования | Лекция      | Корпус А | А-101 | Смирнов Иван Петрович | Friday
```

Процедура для записи слушателя на курс:

```
CREATE OR REPLACE PROCEDURE enroll_student_to_group(
  IN student_id INT,
  IN group_id INT,
  OUT result_message TEXT,
  IN start_date DATE DEFAULT CURRENT_DATE
```

```
)
LANGUAGE plpgsql
AS $$
DECLARE
    end_date DATE;
    group_exists BOOLEAN;
    student_exists BOOLEAN;
    already_enrolled BOOLEAN;
    max_id INT;
BEGIN
    SELECT EXISTS(SELECT 1 FROM "Группа" WHERE "id_группы" =
group_id) INTO group_exists;


    SELECT EXISTS(SELECT 1 FROM "Слушатель" WHERE "id_слушателя" =
student_id) INTO student_exists;


    SELECT EXISTS(
        SELECT 1 FROM "Состав_группы"
        WHERE "id_группы" = group_id AND "id_слушателя" = student_id
    ) INTO already_enrolled;


    IF NOT group_exists THEN
        result_message := 'Группа с указанным ID не существует';
    ELSIF NOT student_exists THEN
        result_message := 'Слушатель с указанным ID не существует';
```

```
    ELSIF already_enrolled THEN
        result_message := 'Слушатель уже записан в эту группу';
    ELSE
        SELECT COALESCE(MAX("id_записи"), 0) INTO max_id FROM
"Состав_группы";

        end_date := start_date + interval '6 months';

        INSERT INTO "Состав_группы" (
            "id_записи",
            "id_группы",
            "id_слушателя",
            "дата_нач_обуч",
            "дата_оконч_обуч"
        ) VALUES (
            max_id + 1,
            group_id,
            student_id,
            start_date,
            end_date
        );

        result_message := 'Слушатель успешно записан на курс';
    END IF;
END;
```

$$;

```
courses_fin=# CALL enroll_student_to_group(1, 2, '2025-05-25', NULL);
          result_message
 ---------------------------------
  Слушатель успешно записан на курс
```

Процедура для получения перечня свободных лекционных аудиторий:

```
CREATE OR REPLACE FUNCTION get_free_lecture_rooms_table(check_date
DATE)

RETURNS TABLE (

    room_number VARCHAR(5),

    building VARCHAR(20)

)

LANGUAGE plpgsql

AS $$

BEGIN

    RETURN QUERY

    SELECT

        a."номер_аудитории"::VARCHAR(5),

        k."название"::VARCHAR(20)

    FROM

        "Аудитория" a

        JOIN "Корпус" k ON a."id_корпуса" = k."id_корпуса"

        JOIN "Тип_аудитории" t ON a."id_тип_аудитории" =
t."id_тип_аудитории"

    WHERE

        t."название_типа" = 'Лекционная'

        AND NOT EXISTS (
```

```
        SELECT 1
        FROM "Занятие" z
        WHERE
            z."номер_аудитории" = a."номер_аудитории"
            AND z."дата_занятия" = check_date
    );
END;
$$;
```

```
 room_number | building
-------------+----------
 A-101       | Корпус A
 B-101       | Корпус B
```

Триггеры

Триггер для проверки дат обучения слушателя:

```
CREATE OR REPLACE FUNCTION check_student_dates()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."дата_оконч_обуч" <= NEW."дата_нач_обуч" THEN
    RAISE EXCEPTION 'Дата окончания обучения должна быть позже даты начала';
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validate_student_dates
BEFORE INSERT OR UPDATE ON "Состав_группы"
FOR EACH ROW EXECUTE FUNCTION check_student_dates();
```

```
courses_fin=# INSERT INTO "Состав_группы" ("id_записи", "id_группы", "id_слушателя", "дата_нач_обуч", "дата_оконч_обуч")
courses_fin-# VALUES (100, 1, 1, '2025-01-10', '2025-01-09');
ERROR:  Дата окончания обучения должна быть позже даты начала
```

Триггер для проверки квалификации преподавателя:

```
CREATE TABLE IF NOT EXISTS "Квалификация_преподавателя" (
   "id_записи" SERIAL PRIMARY KEY,
   "id_препода" INT NOT NULL REFERENCES "Преподаватель"("id_препода"),
   "код_дисциплины" VARCHAR(20) NOT NULL REFERENCES "Дисциплина"("код_дисциплины"),
   "дата_подтверждения" DATE NOT NULL DEFAULT CURRENT_DATE
);

INSERT INTO "Квалификация_преподавателя" ("id_препода", "код_дисциплины")
VALUES (1, 'ДИС-001'), (2, 'ДИС-002');

CREATE OR REPLACE FUNCTION check_teacher_qualification()
RETURNS TRIGGER AS $$
BEGIN
   IF NOT EXISTS (
      SELECT 1 FROM "Квалификация_преподавателя"
      WHERE "id_препода" = NEW."id_препода"
      AND "код_дисциплины" = NEW."код_дисциплины"
   ) THEN
      RAISE EXCEPTION 'Преподаватель % не имеет квалификации для ведения дисциплины %',
         (SELECT "фио" FROM "Преподаватель" WHERE "id_препода" = NEW."id_препода"),
         (SELECT "название" FROM "Дисциплина" WHERE "код_дисциплины" =
NEW."код_дисциплины");
   END IF;
   RETURN NEW;
END;
```

```
$$ LANGUAGE plpgsql;


CREATE TRIGGER validate_teacher_qualification

BEFORE INSERT OR UPDATE ON "Занятие"

FOR EACH ROW EXECUTE FUNCTION check_teacher_qualification();
```

```
courses_fin=# INSERT INTO "Занятие" ("id_занятия", "id_препода", "код_дисциплины", "номер_аудитории", "тип_занятия", "дата
_занятия", "id_группы")
courses_fin-# VALUES (100, 1, 'ДИС-002', 'А-101', 'Лекция', '2025-04-15', 1);
ERROR:  Преподаватель Смирнов Иван Петрович не имеет квалификации для ведения дисциплины Веб-разработка
```

Триггер для проверки занятости аудитории:

```
CREATE OR REPLACE FUNCTION check_room_availability()

RETURNS TRIGGER AS $$

BEGIN

  IF EXISTS (

    SELECT 1 FROM "Занятие"

    WHERE "номер_аудитории" = NEW."номер_аудитории"

    AND "дата_занятия" = NEW."дата_занятия"

    AND "id_занятия" != COALESCE(NEW."id_занятия", -1)

  ) THEN

    RAISE EXCEPTION 'Аудитория % уже занята на эту дату', NEW."номер_аудитории";

  END IF;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;


CREATE TRIGGER validate_room_availability

BEFORE INSERT OR UPDATE ON "Занятие"

FOR EACH ROW EXECUTE FUNCTION check_room_availability();
```

```
courses_fin=# INSERT INTO "Занятие" ("id_занятия", "id_препода", "код_дисциплины", "номер_аудитории", "тип_занятия", "дата
_занятия", "id_группы")
courses_fin-# VALUES (100, 1, 'ДИС-001', 'А-101', 'Лекция', '2025-04-11', 1);
ERROR:  Аудитория А-101 уже занята на эту дату
```

Триггер для автоматического обновления истории должностей:

```
CREATE OR REPLACE FUNCTION update_position_history()

RETURNS TRIGGER AS $$

BEGIN

  UPDATE "История_должностей"

  SET "дата_по" = CURRENT_DATE

  WHERE "id_препода" = NEW."id_препода"

  AND "дата_по" IS NULL;



  RETURN NEW;

END;

$$ LANGUAGE plpgsql;



CREATE TRIGGER track_position_changes

AFTER INSERT ON "История_должностей"

FOR EACH ROW EXECUTE FUNCTION update_position_history();
```

```
courses_fin=# INSERT INTO "История_должностей" ("id_истории_должностей", "id_препода", "id_должности", "дата_с")
courses_fin-# VALUES (100, 1, 2, CURRENT_DATE);
INSERT 0 1
courses_fin=# SELECT "дата_по" FROM "История_должностей"
courses_fin-# WHERE "id_препода" = 1 AND "id_истории_должностей" != 100;
  дата_по
------------
 2025-05-26
(1 row)
```

Триггер для проверки типа аудитории:

```
CREATE OR REPLACE FUNCTION validate_room_type()

RETURNS TRIGGER AS $$

DECLARE

  room_type TEXT;

BEGIN

  SELECT "название_типа" INTO room_type

  FROM "Тип_аудитории"

  WHERE "id_тип_аудитории" = (

    SELECT "id_тип_аудитории" FROM "Аудитория"
```

```
    WHERE "номер_аудитории" = NEW."номер_аудитории"
  );

  IF room_type = 'Компьютерный класс' AND NEW."тип_занятия" = 'Лекция' THEN
    RAISE EXCEPTION 'Лекции нельзя проводить в компьютерных классах';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER check_room_type_compatibility
BEFORE INSERT OR UPDATE ON "Занятие"
FOR EACH ROW EXECUTE FUNCTION validate_room_type();
```

```
courses_fin=# INSERT INTO "Занятие" ("id_занятия", "id_препода", "код_дисциплины", "номер_аудитории", "тип_занятия", "дата
_занятия", "id_группы")
courses_fin-# VALUES (100, 1, 'ДИС-001', 'Б-202', 'Лекция', '2025-04-15', 1);
ERROR:  Лекции нельзя проводить в компьютерных классах
```

Триггер для ведения лога изменений преподавателей:

```
CREATE TABLE IF NOT EXISTS "Преподаватель_лог" (
  "id_записи" SERIAL PRIMARY KEY,
  "id_препода" INT,
  "старое_фио" VARCHAR(60),
  "новое_фио" VARCHAR(60),
  "дата_изменения" TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  "пользователь" VARCHAR(50) DEFAULT current_user
);

CREATE OR REPLACE FUNCTION log_teacher_changes()
RETURNS TRIGGER AS $$
```

```
BEGIN
  IF NEW."фио" != OLD."фио" THEN
    INSERT INTO "Преподаватель_лог"
    ("id_препода", "старое_фио", "новое_фио")
    VALUES (OLD."id_препода", OLD."фио", NEW."фио");
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER track_teacher_updates
AFTER UPDATE ON "Преподаватель"
FOR EACH ROW EXECUTE FUNCTION log_teacher_changes();
```

```
courses_fin=# UPDATE "Преподаватель" SET "фио" = 'Смирнов Иван Петрович (изменено)' WHERE "id_препода" = 1;
UPDATE 1
courses_fin=# SELECT * FROM "Преподаватель_лог" WHERE "id_препода" = 1;
 id_записи | id_препода |      старое_фио       |            новое_фио             |      дата_изменения       | пользоват
ель
-----------+------------+-----------------------+----------------------------------+---------------------------+----------
----
         1 |          1 | Смирнов Иван Петрович | Смирнов Иван Петрович (изменено) | 2025-05-26 03:31:03.96347 | postgres
(1 row)
```

Триггер для проверки email слушателя:

```
CREATE OR REPLACE FUNCTION validate_student_email()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW."электр_почта" IS NOT NULL AND NEW."электр_почта" !~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$' THEN
    RAISE EXCEPTION 'Некорректный формат email: %. Пример правильного формата: student@example.com', NEW."электр_почта";
  END IF;

  RETURN NEW;
```

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check_student_email_format

BEFORE INSERT OR UPDATE OF "электр_почта" ON "Слушатель"

FOR EACH ROW EXECUTE FUNCTION validate_student_email();

```
courses_fin=# INSERT INTO "Слушатель" ("id_слушателя", "ФИО", "паспорт_данные", "электр_почта")
courses_fin-# VALUES (100, 'Тестовый Слушатель', '1234 567890', 'неправильный-email');
ERROR:  Некорректный формат email: неправильный-email. Пример правильного формата: student@example.com
```

Вывод

В ходе выполнения данной лабораторной работы были изучены и практически применены хранимые процедуры, функции и триггеры. Самостоятельная разработка и тестирование этих компонентов на базе реализованной базы данных позволили глубже понять их работу и приобрести навыки создания качественных систем управления базами данных.